Université de Lille

TP 5.09
Spring Security

R5.A.09 - Frameworks WEB
BUT3 - Info
Philippe Mathieu
2024–2025

Objectifs

Utilisation avancée de Spring. Usage de Spring Security.

Préambule

1. Créez un projet SpringBoot via `Spring CLI` (ou le site SpringInitializr) nommé `tp509` avec groupId `fr.but3` et les starters **Web**, **devtools**, **jpa**, **h2** et **Actuator** (on ne met volontairement pas `security` maintenant, on le rajoutera après, à la main)
2. Pour les JSP, modifiez le `.pom` et le `application.properties` pour que l'application accepte les JSP (properties + tomcat-embed-jasper). Vérifiez que le packaging est bien en `war`
3. Créez 2 répertoires `public` et `private` dans `src/main/webapp/WEB-INF/jsp` ainsi qu'une page `public/v1.jsp` avec un message "cette page est publique" et une page `private/v2.jsp` avec un message "cette page est privée".
4. Créez un contrôleur dans `main/java/fr/but3/tp509` nommé `MonControleur.java` avec 2 end-points `private` et `public` qui dispatchent respectivement vers ces deux vues.
5. Testez le bon fonctionnement : il doit être possible d'accéder aux deux pages sans problème.

```
   mvn clean package
mvn spring-boot:run
http://localhost:8080/public
http://localhost:8080/private
```

Exercice 1 : SpringSecurity par défaut

1. Ajoutez uniquement la dépendance **spring-boot-starter-security**.
2. Par défaut Spring security protège **toutes les pages**. Plus rien n'est accessible sans login/mdp . Par défaut le login est `user` et le mot de passe est affiché dans la console au lancement. On constate qu'une page de `logout` est automatiquement gérée. Testez :

```
   mvn clean package
mvn spring-boot:run
http://localhost:8080/public
http://localhost:8080/logout
http://localhost:8080/private
http://localhost:8080/logout
```

3. Ajoutez maintenant les 2 propriétés `spring.security.user.name` et `spring.security.user.passwo`
   Voir T15 du cours

4. Retestez avec ce nouveau login/mdp.
   Vous remarquerez que plus aucun password n'est affiché dans la console et que l'ancien login/mdp ne fonctionne plus.
5. On souhaite afficher sur la vue `v2.jsp` le nom de l'utilisateur connecté. Testez deux possibilités :
   (a) Ajoutez dans les 2 vues le nom de la personne connectée en injectant `Principal` dans les méthodes du controleur
   (b) Utilisez directement dans la vue `getRemoteUser()` sur l'objet `request`.

Exercice 2 : SpringSecurity paramétré

1. Mettre en place une configuration de sécurité qui autorise l'accès à `public` sans login/mdp et l'accès à `private` avec un login/mdp différent de l'exercice précédent.
   Les login/mdp seront gérés en mémoire.
   Le hashage sera réalisé en BCrypt
   La bannière de login sera gérée par le navigateur.

   Exemple minimal à adapter :

```
@Configuration
@EnableWebSecurity
public class Security {
    @Bean
    public SecurityFilterChain mesautorisations(HttpSecurity http, HandlerMappingIntrospector intr
    throws Exception {
        MvcRequestMatcher.Builder mvc = new MvcRequestMatcher.Builder(introspector);
        return http.authorizeHttpRequests((authorize) -> authorize
            .anyRequest().authenticated()
        )
        .httpBasic( Customizer.withDefaults() )
        .build();
    }

    @Bean
    public UserDetailsService mesutilisateurs() {
        UserDetails user1 = User.withUsername("user")
            .password(encoder().encode("sonpassword"))
            .roles("ADMIN")
            .build();
            return new InMemoryUserDetailsManager(user1);
    }

    @Bean
    public PasswordEncoder encoder() {
            return new BCryptPasswordEncoder();
    }
}
```

2. Testez l'accès à `public` puis testez l'accès à `private` comme précédemment. En `HttpBasic` un token `Basic` est envoyé à chaque requête par le navigateur. Pour se déconnecter, il faut quitter le navigateur.
3. Remplacez `httpBasic()` par `formLogin()`. Cette fois Spring fournit une page de login. Le token est placé en session. Vérifiez dans votre navigateur (outils de dev Web / Stockage / cookies)
4. Avec `formLogin()` vous avez en prime le endpoint `/logout` pour la déconnexion.

5. Nous avons encodé un mot de passe en clair dans le code, ce qui n'est pas conseillé. Hachez votre mdp en BCrypt et fournissez le tel quel lors de la définition de l'utilisateur.
   Il existe de nombreux sites et outils permettant de faire cela, `spring cli` en est un.
   Tapez `spring help encodepassword`
6. Modifiez votre configuration pour que `/logout` amène à la page `/public`
7. Rajoutez la gestion du `rememberMe` . Une checkbox est apparue. Vous constatez que vous avez maintenant deux cookies. Si vous supprimez le cookie de session, vous êtes encore connecté.

Exercice 3 : Authentification via JDBC

Modifiez votre `SecurityFilterChain` (composant de sécurité) pour permettre une authentification via JDBC sur une base de données.

Q1. Pour cela suivez les étapes suivantes

1. La dépendance `spring-jdbc` (ou `spring-jpa`) dans le pom
2. Le paramétrage de la base de données dans `application.properties`
3. La création des 2 tables. `Spring JPA` lance à chaque démarrage un fichier `schema.sql` et `data.sql`. Ecrire le script SQL de création des tables dans `resources/schema.sql`

```sql
-- schema.sql
DROP TABLE IF EXISTS authorities CASCADE;
DROP TABLE IF EXISTS users CASCADE;

CREATE TABLE users
(
username VARCHAR(50)  NOT NULL,
password VARCHAR(100) NOT NULL,
enabled  BOOLEAN      NOT NULL DEFAULT TRUE,
PRIMARY KEY (username)
);

CREATE TABLE authorities
(
username  VARCHAR(50) NOT NULL,
authority VARCHAR(50) NOT NULL,
FOREIGN KEY (username) REFERENCES users (username)
);
```

4. Modification du bean `UserDetailService` pour qu'il soit instancié avec un `JdbcUserDetailManager`

Q2. Testez à nouveau l'application dans tous les sens. Affichez la table `users` (voir slide 29)

Exercice 4 : L'encodeur par défaut : DelegatingPasswordEncoder

1. Jusqu'à présent nous avons utilisé notre propre encodeur. Spring en possède un par défaut : le `DelegatingPasswordEncoder`. Celui ci est capable de gérer tous les encodages, moyennant un préfixe. D'un côté il réalise le hash en `BCrypt`, de l'autre côté il sait vérifier les hachages standard (dont `bcrypt`, `scrypt`, `md5` ou `noop`).
   Remplacez le précédent encodeur par un `DelegatingPasswordEncoder`. Ajoutez aussi un fichier `data.sql` avec deux utilisateurs dedans.

```sql
-- data.sql
INSERT INTO users VALUES ('paul','{MD5}6c63212ab48e8401eaf6b59b95d816a9',TRUE);
INSERT INTO users VALUES ('pierre','{noop}pierre',TRUE);

INSERT INTO authorities VALUES ('paul', 'USER');
INSERT INTO authorities VALUES ('pierre', 'USER');
```

2. Vérifiez la manière dont les mots de passe sont écrits dans la base
3. Testez les login (paul,paul), (pierre,pierre), ainsi que celui que vous avez créé dans le code.
4. Spring conserve les informations de la personne connectée dans un objet `Principal`. Ajoutez à votre contrôleur un endpoint `/principal` qui renvoie l'objet `Principal` en JSON.

Q1. À l'aide de votre IDE, demandez la définition du `DelegatingPasswordEncoder`. Visualisez la liste des décodeurs disponibles.

Exercice 5 : Création de nouveaux utilisateurs

Jusqu'à présent les utilisateurs créés ont été mis "à la main", soit en SQL (dans le schema.sql) soit dans le code (avec `.withUser`). On souhaite maintenant pouvoir créer des utilisateurs par programmation.

1. Mettre en place un bean `JdbcUserDetailsManager`
2. Utilisez ce bean pour enregistrer un nouveau `UserDetails`
3. Testez la création d'un utilisateur directement dans la classe `WebSecurityConfig`.
4. Mettre en place un endPoint `/creer` qui lit deux paramètres HTTP login/mdp et qui crée l'utilisateur correspondant.